CONF-9509237-

LA-UR- 95-3298

Title:

Shlaer-Mellor Object-Oriented Analysis and Recursive Design, an Effective Modern Software Development Method for Development of Computing Systems for a Large Physics Detector

Author(s):

T. Kozlowski, T. A. Carey, C. F. Maguire, D. Whitehouse, C. Witzig, and S. Sorensen

Submitted to:

Conference on Computing in High Energy Physics
Rio de Janeiro, Brazil, September 18–22, 1995

MASTER

Los Alamos
NATIONAL LABORATORY

Form No 836 R5
ST 2629 10/91

SHLAER-MELLOR OBJECT-ORIENTED ANALYIS AND RECURSIVE DESIGN, AN EFFECTIVE MODERN SOFTWARE DEVELOPMENT METHOD FOR DEVELOPMENT OF COMPUTING SYSTEMS FOR A LARGE PHYSICS DETECTOR

THOMAS KOZLOWSKI, THOMAS A. CAREY

Physics Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

CHARLIE F. MAGUIRE

Physics Department, Vanderbilt University, Nashville, TN 37235, USA

DAVID WHITEHOUSE, CHRIS WITZIG

Physics Department, Brookhaven National Laboratory, Upton, NY 11973, USA

SOREN SORENSEN

Department of Physics, University of Tennessee, Knoxville, TN 37996-1200, USA

After evaluation of several modern object-oriented methods for development of the computing systems for the PHENIX detector at RHIC, we selected the Shlaer-Mellor Object-Oriented Analysis and Recursive Design method as the most appropriate for the needs and development environment of a large nuclear or high energy physics detector. This paper discusses our specific needs and environment, our method selection criteria, and major features and components of the Shlaer-Mellor method.

1 PHENIX Detector Software Development Requirements

1.1 Scope and Environment of PHENIX Computing

The PHENIX detector under construction at the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory is typical in size and environment of other large collider detectors. An important aspect is the ten subdetectors under development by different groups within the collaboration that must be integrated (hardware and software) into the frameworks provided by core on-line and offline systems.

Detector up-time requirements and the long lifetime of the project require high quality and reliable software that meets functional and performance requirements for detector operation, is relatively problem free at the outset, and is maintainable over the lifetime of the project. The development process and methods adopted must support a large computing effort, widely distributed geographically, and accommodate integration of contributions from subsystem groups and the use of imported software.

1.2 *Use of a Formal Method*

The need for reliability, long term support, and integration of distributed development activities led us to require a relatively formal development process that incorporates defined work products, regular reviews and careful documentation. Rather than trying to develop our own processes and methods, it was decided to survey and take advantage of modern software engineering technologies. We expect that relying on the experience of professionals in the software development industry well best help us meet development requirements and maximize the productivity of the limited resources devoted to PHENIX computing.

The software engineering community has long recognized that the efforts applied to requirements analysis and design are important and have great leverage on the efforts and costs of subsequent efforts in implementation and maintenance. Therefore, selecting a formal method of analysis and design was a high priority goal.

2 Modern Analysis and Design Methods

Driven by the need to increase productivity and software reliability, the software engineering community and the software development industry have developed and used many methods for developing software. Drawing on lessons learned and advances in computing science and computing hardware, these methods have been continually evolved. Some recent areas of interest and activity are the total development process, object-orientation, and formal methods and notations in support of analysis and design. Automation of elements of the development process has been a constant theme, and it is now becoming a reality with the advent of modern high powered workstations in combination with more rigorous and complete methods and notations.

In the last decade there has been much research and development in object-oriented analysis and design methods, and the discipline is maturing rapidly. Several methods for object-oriented analysis and design are in wide use on real projects by major software developers. Three of the most popular that are appropriate for large real-time systems are: Booch Object-Oriented Analysis and Design [1], Rumbaugh Object Modeling Technique (OMT) [2], and Shlaer-Mellor Object-Oriented Analysis and Recursive Design [3,4]. Recently the originators of the Booch and OMT methods have combined resources and plan to develop a new method combining features from both predecessors. There is a wide range of CASE tool support for all popular methods.

3 Selection of an Analysis and Design Method

3.1 *Selection Criteria and Constraints*

Limited time and resources did not allow us to do an exhaustive survey and trial use of development methods. We committed ourselves to a relatively rapid and informal survey of major current development methods, relying on the opinions of others and our own experiences (and prejudices) in coming up with a preliminary selection. After a period of initial use our selection will be re-evaluated.

Table 1: **Estimated Conformance of Evaluated Methods with Selection Criteria.**

Criterion	Booch	Shlaer-Mellor	OMT
Real-time and concurrency	medium	high	medium
Partitioning	medium	high	high
Completeness and consistency	low	high	low
Doesn't assume OO implementation	low	high	low
Accommodates imported software	medium	high	medium
Compatible with iterative development	high	medium	high
Widely and successfully used	high	high	high
Good CASE support exists	high	high	high

Our analysis and design method selection criteria included the following:

- support of object-oriented analysis, design, and implementation
- explicit support for modeling real-time applications and concurrency
- effective partitioning of large systems for concurrent development
- specific criteria for completeness and consistency
- design or implementation not constrained to be object-oriented
- accommodation of imported software and software systems
- compatible with an iterative development process
- widely and successfully used
- good CASE support

3.2 Selection of Shlaer-Mellor OOA and Recursive Design

After studying the literature and Usenet newsgroups, attending vendor presentations, drawing on our own experiences in developing software, and discussing features and issues among ourselves, we provisionally selected the Shlaer-Mellor method. This was admittedly an informal and somewhat subjective process. Table 1 summarizes our results in applying our criteria to the three methods.

We believe the Shlaer-Mellor method best meets our needs and environment. We found some aspects of the method particularly attractive:

- The explicit partitioning, vertically into subject matter domains and horizontally within domains into subsystems, simplifies concurrent and distributed development and permits effective use of expertise in particular subject matters within the group. The explicit separation of the application domain from the service, architecture and implementation domains is useful because that while in our case the application domain (detector and data acquisition hardware, and operations) is relatively stable over the lifetime of the object, the project can benefit over its lifetime from new and evolving technologies for service, architecture, and implementation domains. On the other hand, if there is a major change in the detector, it is likely that only the OOA model of the application domain need be changed and the corresponding software re-generated using the same architecture as before.

- The fact that the method explicitly includes well defined process steps and work products, and rigorous rules and criteria for completeness and consistency allow us to leverage off the considerable experience of the method's developers; otherwise, we would have to develop our own process to an equivalent level of detail and refine it through use. There is anecdotal evidence in the literature that developers that modified the Shlaer-Mellor process for their own needs usually ended up re-inventing the original process. The rigor and completeness of the models is a foundation for simulation via actual "execution" of the models and automated code generation.

- Shlaer-Mellor is widely used by many important developers of real-time and process control applications. The originators and their company (Project Technology) have a long history as consulants, trainers, and contractors in development of real-time software systems. They actually use their method on a wide variety of projects.

A negative aspect of the Shlaer-Mellor method is that the simple and limited notation (for example, the lack of aggregation or composite objects in the formalism) and rules, often lead to large and complex models that are hard to comprehend. However, there is a trade off here, since in the case of a more complex notation a much more complex set of rules (and probably extensions to the notation) would be required to guarantee completeness and consistency. This rigor is an important advantage of the method.

The Shlaer-Mellor method is not the best foundation for iterative development in the style of "rapid prototyping", because of the rigor and completeness required for OOA models and the associated the effort required. The method is more suited to a development style in which there are widely spaced releases, each of which is a full implementation of a subset of the final system, perhaps with prototype service and architectural domains in the earlier releases.

3.3 Support

The originators of the method, Sally Shlaer and Stephen Mellor, founded a company in 1985, Project Technology, which carries on their work. The company continues development and refinement of the method, along with consulting and method training.

Several major CASE vendors provide relatively complete support for Shlaer-Mellor. Some provide or soon will provide simulation and code generation tools. Project Technology itself has recently acquired a CASE tool vendor (Bridgepoint). We have provisionally selected the ObjectTeam tool set from Cadre Technologies, Inc., and are currently evaluating the Bridgepoint tool set.

4 Overview of Shlaer-Mellor OOA and Recursive Design

The rational behind the method is well stated in the goal of Project Technology: "a systematic, rational and controllable software development process". The method has developed and evolved over the years through from experience in building real-time software systems. It is based on the central role and value of an "information model" (the basis of its object-orientation).

4.1 Domains and Subsystems

The first step in creating Shlaer-Mellor models is to partition the system into "orthogonal" subject matter domains. A domain is "a real, hypothetical, or abstract world inhabited by a distinct set of objects that behave according to rules and policies characteristic of the domain" [4]. Domains can be of the following types:

- Application Domain: subject matter of the system from the perspective of the user of the system.
- Service Domain: provides generic mechanisms and utility functions as required to support the application domain (and other service domains).
- Architectural Domain: defines a "mapping" of OOA model elements to implementation domain elements, and provides generic mechanisms and structures for managing data and control for the system as a whole.
- Implementation Domain: languages, networks, operating systems, libraries.

Domains may be developed according to Shlaer-Mellor OOA, implemented by "conventional" (non-Shlaer-Mellor) methods, or may be imported software. The work products of domain analysis are a graphical domain chart that shows the domains and the client/server dependency relationships between them ("bridges"), and the accompanying textual descriptions of each domain and bridge. An example of a domain chart for a simple data acquisition application is shown in Figure 1. The application domain is at the top, and the implementation domains are at the bottom.

Analysis of domains begins by identifying the objects that make up a domain's subject matter. The method permits large domains to be partitioned into "subsystems" which are clusters of related objects with minimal relations with other subsystems. A subsystem is the scope for most of the remaining modeling activities.

4.2 Fundamental Analysis Models

The method specifies the system requirements of a domain by formally defining the conceptual entities (objects), their attributes and relations, their behavior (lifecycle), and the operations they can perform. Note that the Shaler-Mellor method uses the word "object" to designate an abstract type or "class", and the word "instance" to designate a particular instantiation. Three fundamental types of models for analysis are defined: information, state and process models.

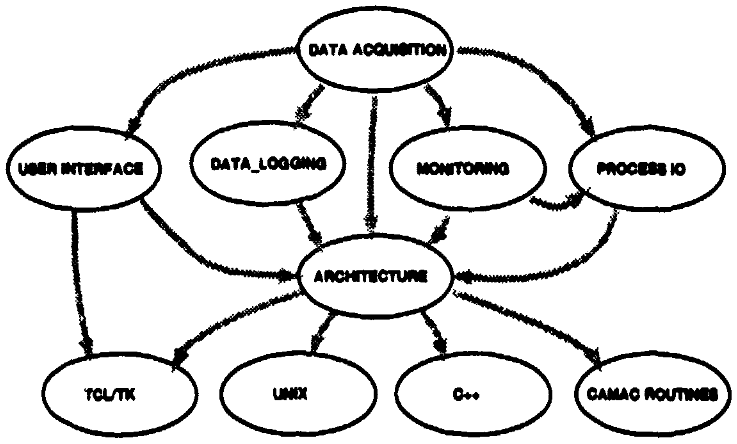

Figure 1: Example of a Domain Chart

- Information Model:

  For each subsystem an information model is created. It consists of a graphical diagram describing subsystem objects, their relations and their attributes, and accompanying formal textual descriptions of all objects, relations and their attributes. The object diagram is similar to an Entity Relationship Diagram (ERD). Attributes must be single variables (not structures); an object must contain identifier attributes that uniquely identify an instance; and relations are formalized by referential attributes. Supertype and subtype objects ("is a" relationships) are part of the formalism. For each instance of a subtype object there must be an instance of a supertype object and *vice versa*. Associative objects are used to model many-to-many relationships and also relationships that may have attributes associated with them.

- State Models:

  For each object that is identified as having a lifecycle ("active object"), a state model is created using the Moore formalism (actions only occur on entering a state). It consists of a graphical state transition diagram describing states, transitions, events that cause transitions, actions occuring on entering states, and the accompanying formal textual descriptions of events and actions. Events provide a communication mechanism between objects, and between objects and the outside world. Data can be associated with an event. Actions can do calculations, create and delete instances, read and write attributes, and generate events. The method includes a complete set of time rules and event handling rules. Examples of some rules are: actions are atomic; multiple state machines (instances of an object) can execute simultaneously; events are never lost; an event is used up when accepted by a state machine. The method includes a special "assigner state machine" that can be associated with an object (usually an associative object) as a standard way to model "competitive" relationships (for example, assigning a resource to a requester). Timer objects are included in the formalism to explicitly handle time constraints.

- Process Models:

  For each state that corresponds to a non-trivial action, an action dataflow diagram (ADFD) is created. Process bubbles correspond to atomic processing steps. The only allowed dataflow labels are those identified with event data or object attributes. Stores must be identified with objects. A process model consists of a ADFD and accompanying formal textual descriptions of the processes. Processes must be one of five types: accessors, event generators, transformations, tests, bridges (to other domains). Object attribute "accessor processes" are a data communication mechanism between objects. Each process is assigned to one object in the information model (for example, an accessor object is assigned to the object which it accesses). Many CASE tools omit the more general process model, and instead rely on the state model action descriptions to formally specify the processing.

- Derived models:

  The method defines several models that are derivable from the fundamental models but provide specific views of the system model that are useful to an analyst. At the domain level these are:

  - Subsystem Relationship Model: relations between subsystems
  - Subsystem Communication Model: event communication between subsystems
  - Subsystem Access Model: data (object attribute) access between subsystems

  At the subsystem level these are:

  - Object Communication Model: event communication between objects
  - Object Access Model: data (object attribute) access between objects

4.3 Recursive Design

Design in the Shlaer-Mellor method resides in the choices for implementatior of the service and architectural domains. Recursive design refers to repeatedly applying the OOA process to the application, service, and architectural domains. In most implementations, OOA is probably not applied to all domains since some may be imported or developed by conventional methods. The ideal sequence is to first apply OOA to the application domain. This analysis determines requirements on the application's service domains. OOA is then applied to service domains in an appropriate order. Additional requiements on service domains may be determined through this analysis. Performance requirements and constraints from software and hardware environments determine the architectural domain requirements. The architectural domain can also be analyzed using OOA. Once the architectural domain

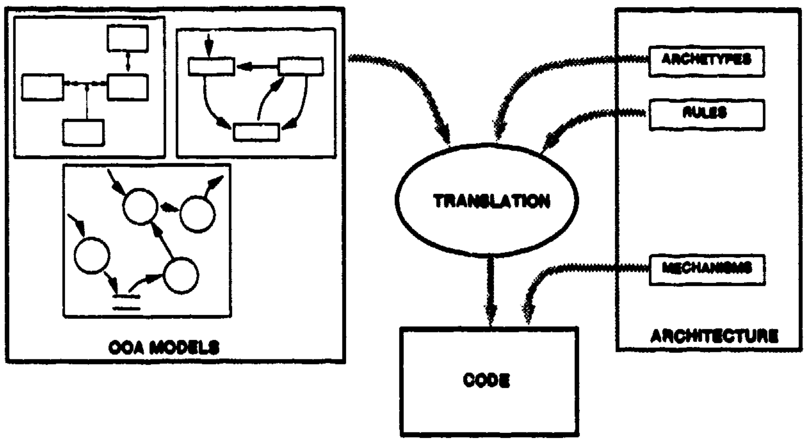

Figure 2: Translation of OOA Models.

is implemented, it is applied (in reverse order) to the service and application domains. The result is an implementation of the system. In practice, because of prior experience and existing knowledge in the development team, domains are usually developed in parallel.

4.4 *Architecture Domain*

In most methods a design and implementation is developed by adding implementation layer elements to the application models, without a clear or formal separation of the layers. There is really only one model of the system. Shlaer-Mellor proponents refer to this as "implementation by elaboration". The Shlaer-Mellor method, via the architecture, translates the OOA models into the implementation. This is referred to as "implementation by translation". The cost of implementation using translation, since it depends on the cost of a relatively fixed and independent architecture, can be much less than the cost of implementation using elaboration which tends to be proportional to the size of the analysis model. In addition, one can see that this paradigm lends itself well to automated translation (code generation). Such capabilities are beginning to appear in CASE tools that support the Shlaer-Mellor method. This clear separation between analysis and design is a unique feature of the Shlaer-Mellor method. The analysis model of the application domain or a service domain is independent of the architecture(s) that will be developed. In a few words the method permits building of "implementation free" applications (OOA models) and "application free" implementations (architectures). This promotes the reuse and independent evolution of both applications and architectures. The relation between OOA models, architecture, and implementation is shown schematically in Figure 2.

The components of an architecture have three forms:

- Mechanisms: code that can be linked into the final system that implements elements of the formalism (state machines, event receiving queues, etc.)

- Archetypes: templates for code fragments that are filled in ("populated") based on elements in the OOA model (for example, there may be archetypes for C++ classes that have a direct relation to objects in the OOA model)
- Translation rules: describe how the archetypes are to be filled

The exact form and complexity of an architecture depends on the implementation environment. Architectures for multi-tasking multiple processor environments and which use much imported software will be more complex and larger than architectures for single task, single processor, self-contained C++ environments. There is no requirement that implementation languages be object-oriented. If they are not, the architecture takes care of the mapping from the OOA models to the elements of non-object-oriented implementation environment.

4.5 Some Imvlementation Issues

The independent architectural domain allows developers to make efficient use of personnel and their expertise. Most developers do not have to concern themselves with implementation issues at all; on the other hand, those who are expert in operating system, language and networking technologies can apply that expertise in the architectural domain, without requiring a detailed knowledge of the application.

The application independent nature of architectures allows whole architectures to be reused with minimal modification, although some tayloring to specific environments is probably always needed. It is an opportunity for vendors to market architectures. Project Technology has begun to do this. "Off the shelf" architectures, together with automated code generation tools from CASE vendors, could lead to a very productive code develoment process.

Another paper in this conference [5] describes the specific application of the Shlaer-Mellor method to PHENIX on-line computing. Our plans an· experiences will be discussed in more detail there.

Acknowledgments

This work has been funded by the U.S. Department of Energy.

References

1. Grady Booch, ***Object Oriented Analysis and Design***, Second edition (The Benjamin/Cummings Publishing Company, 1994).
2. James Rumbaugh *et al*, *Object-Oriented Modeling and Design* (Prentice Hall, 1991).
3. Sally Shlaer and Stephen J. Mellor, *Object Oriented Systems Analysis, Modeling the World in Data* (Yourdon Press, 1988).
4. Sally Shlaer and Step' en J. Mellor, *Object Lifecycles, Modeling the World in States* (Yourdon Press, 1992).
5. Chris Witzig, *et al*, "The application of Shlaer-Mellor Object-Oriented Analyis and Recursive Design to PHENIX On-Line Computing Systems" in the proceedings of this conference.